NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A LEARNING STRATEGY APPROACH
FOR TEACHING
NOVICE COMPUTER PROGRAMMERS

by

Donald D. Begley

September 1984

Thesis Advisor: Gordon H. Bradley

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: A Learning Strategy Approach for Teaching Novice Computer Programmers	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis September 1984
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Donald D. Begley	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School Monterey, CA 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School Monterey, CA 93943	12. REPORT DATE: September 1984
	13. NUMBER OF PAGES: 84
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Learning Strategies; Software Engineering; Advance Organizers; Mnemonics; Elaboration; Cognitive Psychology

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The purpose of this thesis is to investigate various learning strategies and present some suggested applications for the teaching of computer programming to Marine Corps entry-level programmers. These learning strategies are used to develop a cognitively designed structure for the teaching of the software engineering process. This structure was designed so that programmers could have readily available in their thinking

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

process modern software engineering goals and principles that ultimately affect the quality of software.

Also suggested at a lower level of the overall structure is a syntax and semantics organizer. This particular framework serves as an advance organizer for which specific programming language features could be introduced. This structure can act as an organizing mechanism for the introduction of various, useful programming chunks that would start the novice programmer on his quest to becoming an expert.

A Learning Strategy Approach for Teaching Novice Computer Programmers

by

Donald D. Begley
Captain, United States Marine Corps
B.S., Miami University, Ohio

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1984

ABSTRACT

The purpose of this thesis is to investigate various learning strategies and present some suggested applications for the teaching of computer programming to Marine Corps entry-level programmers. These learning strategies are used to develop a cognitively designed structure for the teaching of the software engineering process. This structure was designed so that programmers could have readily available in their thinking process modern software engineering goals and principles that ultimately affect the quality of software.

Also suggested at a lower level of the overall structure is a syntax and semantics organizer. This particular framework serves as an advance organizer for which specific programming language features could be introduced. This structure can act as an organizing mechanism for the introduction of various, useful programming chunks that would start the novice programmer on his quest to becoming an expert.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. MOTIVATION

The major portion of a daily schedule for most military units is set aside for training. In tactical units, the content of the training is rather straightforward; that is, concentration on small unit maneuvers with large-scale war games being the culmination of the effort. But what about the high technology support areas? What kind of training should say, a computer programmer receive initially and how should that expertise be kept current or updated?

Ideally, the computer programmer should be taught not only the command of the predominant programming language in use, but he should also be exposed to some software engineering principles and programming practices that will facilitate the intelligent use of that language. However, the realization that the time and cost of the training program are to be kept to a minimum should not be overlooked, nor should the mastery of the subject area be sacrificed. In short, the goal is to " train 'em well" but "train 'em fast."

Tc accomplish this goal, techniques need to be developed to help the student assimilate the information quickly and store it away in memory efficiently. The issue is not so much what to teach, but how to present the information and how to get the student to recall it when needed in the future.

The purpose of this thesis is to investigate various learning strategies and present some suggested applications for the teaching of computer programming to novice programmers. This learning strategy approach will be designed to take advantage of man's natural cognitive processes, thus enhancing the student's assimilation and recall of information.

The target of the instruction will be a typical class of students that a Marine Corps data processing instructor may encounter in a programming course. This thesis does not address the area of follow-on training and considers the approach to that problem an entirely different research area. Normally, this class of students consists of high school graduates on their first assignment after completion of basic military instruction or "boot camp." The majority of the students have little or no computer background, but, are well-disciplined and highly motivated for learning.

They all now possess a strong, common military background rooted in the areas of marksmanship, close-order drill, small-unit tactics, military customs and courtesies, and physical fitness. Because these students come from such varying backgrounds, their military training may serve as the only common base of experience on which to build new knowledge. This thesis suggests that these recent, varied military experiences can be used as a solid foundation on which to build concrete models, analogies, and other learning strategies for the presentation of difficult programming concepts to novices. Research in cognitive psychology tends to support this approach for enhancing the learning process. This thesis, therefore, proposes that by integrating learning strategies and new concepts based on an already existing knowledge structure, more effective programmers can be developed in a shorter amount of time.

B. INSPIRATION

The subject of this thesis was influenced by the relatively recent merging of theories and methodologies from the fields of cognitive psychology and computer science. The most influential factor contributing to the selection of

this subject area is the employment of some learning techniques by the U. S. Marine Corps in certain crucial areas of training. These simple, but highly effective, training techniques can be shown to conform to certain theories in cognitive psychology, and, in this thesis, some are proposed to be developed in other areas of training, such as computer programming.

C. BACKGROUND

As an example of one of the learning techniques mentioned above, consider the simple use of the word "brass." In programming language jargon, this word may be said to be overloaded -- Marines spend a considerable amount of time in boot camp shining their brass; no marksmanship training is ever complete without a thorough "policing" of the brass (casings) after firing; and every officer knows that he is sometimes "affectionately" referred to as "the brass."

However, while this degree of overloading might lead to some unacceptable ambiguities in programming languages, by the further overloading of the word brass, this time as a memory aid, a highly effective learning technique emerges.

Every Marine who qualifies or requalifies with the rifle uses it. That is, "brass" is also used as an acronym that acts as a mental trigger for the recall of marksmanship principles:

B - Breathe
R - Relax
A - Aim
S - Slack
S - Squeeze

The effectiveness of this technique is rooted in the basic human cognitive processes and can be attributed to what the literature calls, "chunking." But the primary reason for its proven success is that the marksmanship instructors explicitly teach "BRASS" as the organizing tool for the marksmanship principles. Before every round fired, the shooter is taught to concentrate on his "BRASS," and two weeks of marksmanship training is instinctively available for efficient recall.

D. ORGANIZATION

While certainly not all subjects and learning objectives possess the inherently favorable conditions that permit a powerful chunk to be developed, taught, and represented as a

meaningful acronym such as "BRASS," other learning strategies exist that may enhance the learning process in these areas. After a discussion on the role of cognitive psychology in this learning process, several learning strategies will be presented. Another chapter will present some "chunking" ideas that, hopefully, may do for software development what "BRASS" does for marksmanship. Finally, the last chapter deals with possible future research in computer programmer education.

II. THE RCLE OF COGNITIVE PSYCHOLOGY

Cognitive psychology is the study of the mental processes by which sensory input is transformed, reduced, elaborated, stored, recovered, and used. Some commonly known terms that fall under the purview of cognitive psychology are sensation, perception, imagery, retention, recall, problem-solving, and thinking. [Ref. 1]

To help accomplish the goal of training programmers in the Marine Corps effectively and efficiently, this chapter reviews some of the pertinent ideas from the field of cognitive psychology that might help in the area of teaching computer programming to novices. Since the majority of entry level Marine programmer trainees have no previous experience in computer programming, the primary problem is how to present new and difficult technical information in a manner that maximizes meaningful learning while maintaining a minimum training cycle. Thus, the investigation of the learning process should shed some light on the presentation of learning strategies that programming instructors could include in their training materials.

A. MEANINGFUL LEARNING

Cognitive psychologists usually define meaningful learning as the process in which the learner connects new material with the knowledge that is already stored in memory [Ref. 2]. The existing knowledge structures in memory are classified as schema or frames, terminology and ideas that have, incidentally, greatly influenced the artificial intelligence branch of computer science. That process whereby new information is somehow integrated into the existing knowledge structures, or schema, is referred to as "assimilation" [Ref. 3].

Although there is no widespread accepted theory concerning the cognitive mechanisms involved in the assimilation of new material into existing schemas, the following chapter of this thesis presents some suggested learning strategies designed to speed up the assimilation process while, at the same time, enhancing the long-term memory recall ability and retention of novices. In much the same way as the Marine marksman has the chunk named "BRASS" available to remind him of basic marksmanship principles, the Marine programmer will have chunks instinctively available to aid him in the correct development of software.

B. THE HUMAN PROCESSOR MODEL

Card, Moran, and Newell [Ref. 4] present a model of the mental processes relating their interactions to a computer information system. They depict this "Model Human Processor" as three interactive subsystems: (1) the perceptual system, (2) the motor system, and (3) the cognitive system. Each system is complete with its own memory and processors (see Figure 2.1) and interact through ten "principles of operations."

Four of these "principles of operations" are of particular interest in the context of this thesis. These include:

1. Recognize-Act Cycle of the Cognitive Processor. On each cycle of the cognitive processor, the contents of short-term memory initiate actions associatively linked to them in long-term memory.
2. Encoding Specificity Principle. Specific encoding operations on what is perceived determine what is stored, and what is stored determine what retrieval cues are effective in providing access to what is stored.

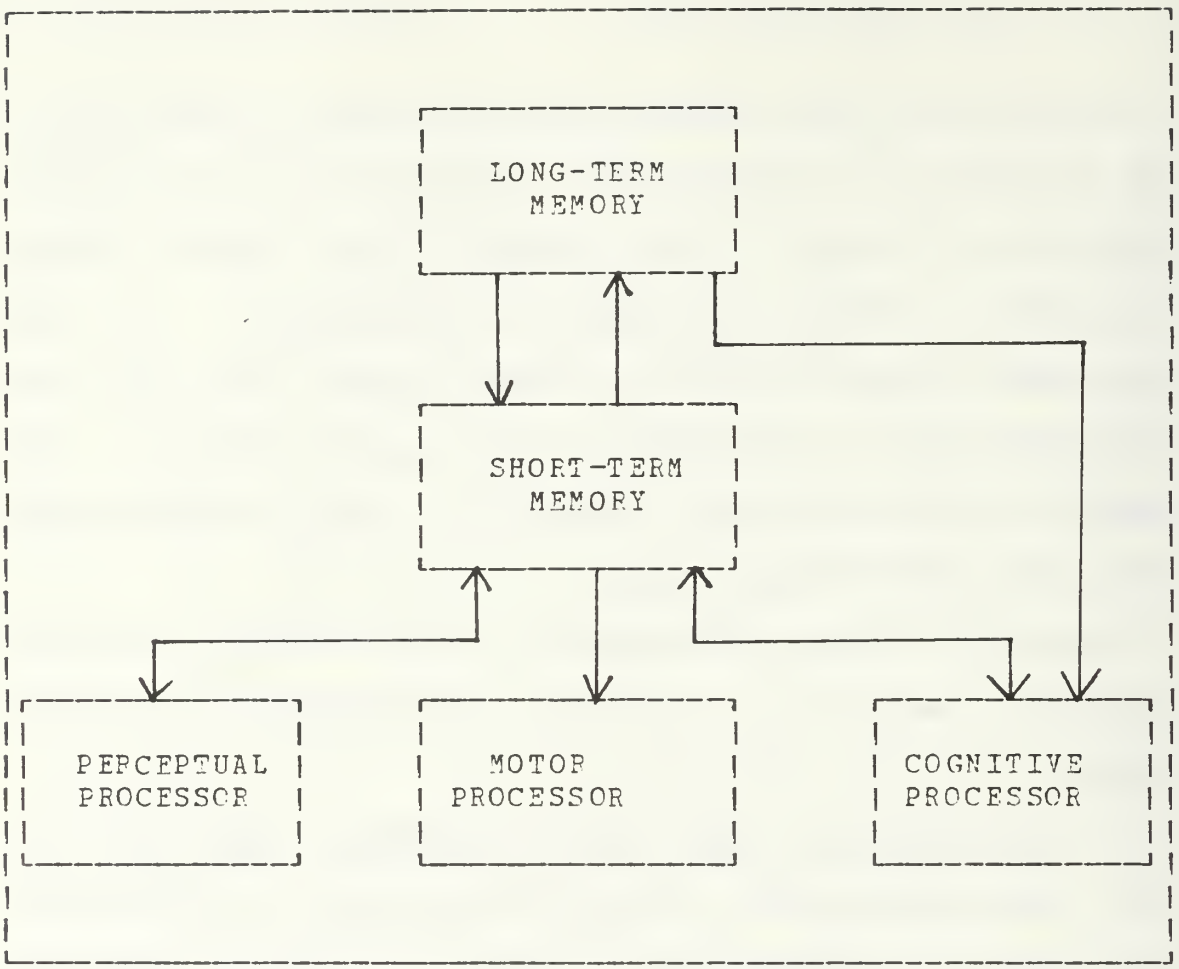

Figure 2.1 The Model Human Processor.

3. Discrimination Principle. The difficulty of memory retrieval is determined by the candidates that exist in the memory, relative to the retrieval cues.
4. Variable Cognitive Processor Rate Principle. The cognitive processor cycle time is shorter when

greater effort is induced by increased task demands of information loads; it also diminishes with practice. [Ref. 5]

The remaining six principles deal mostly with the motor system, decision making, and problem solving and are not considered in this thesis.

To achieve the desired state of meaningful learning as defined earlier, three necessary conditions must be satisfied. Once new technical information enters the system from the environment through the perceptual processor, the conditions that must be met are:

1. Reception. First the learner must pay attention to the incoming information so that it reaches short-term memory.
2. Availability. Next, the learner must possess appropriate prerequisite concepts in long term memory to use in assimilating the new information.
3. Activation. Finally, the learner must actively use this prerequisite knowledge during learning so that the new material may be connected with it. [Ref. 6]

As mentioned earlier, a widely accepted theory on the mechanisms involved in the assimilation of new information does not yet exist. However, Mayer's description [Ref. 7]

of the conditions for meaningful learning and Card's et al. "Model Humar Processor," with its associated principles of operation, combine to form a compatible and workable theoretical foundation for a cognitive approach to teaching Marine Corps novice programmers. The principles of operation, the "Model Human Processor," and the conditions for meaningful learning can be shown to be complementary.

For example, the definition of the Encoding Specificity Principle encompasses both the Reception and Availability conditions. The Recognize-Act Cycle of the cognitive processor provides the necessary information to satisfy the Activation condition. The system acts as a whole when the learner brings new material into short-term memory via the perceptual processor in accordance with the Encoding Specificity Principle. The learner then searches long-term memory for the appropriate prerequisite, or anchoring, ideas and then must transfer these ideas into short-term memory for integration with new incoming information using the cognitive processor [Ref. 8]. This process is controlled primarily by the Recognize-Act Cycle of the cognitive processor and subsumes the Activation condition. While long-term storage is thought to be of infinite capacity, the bottleneck in the above system is short-term memory's

limited capacity to handle only seven, plus or minus two, chunks of information at a time [Ref. 9].

With the reception condition being satisfied based on the assumption that the Marine is highly motivated for learning and the availability condition being satisfied based on common military training being used for appropriate anchoring ideas, it would appear that the activation process and maximizing use of short-term memory's capability might offer promising avenues of research for the enhancement of meaningful learning.

C. CHUNKING

One area of research that might lead to the enhancement of the activation process concerns the manner in which information is stored in long-term memory. Information is thought to be organized into related units called "chunks." These chunks can be hierarchically organized into still larger chunks. The activation of a particular chunk may, in a sense, recursively activate other chunks. When these chunks of information are further associated with familiar acronyms, short-term memory's capability for information retrieval from long-term memory is significantly enhanced.

For example, the following chunk of nine letters would be very difficult to remember without a considerable amount of time devoted to rote memorization -- B C S B M I C P A. However, by rearranging the letters into three recognizable acronyms, CBS RCA IBM, long term retention and efficient recall of the whole chunk of nine letters is a simple task [Ref. 10]. Presuming, of course, that the learner is familiar with some large corporations in the United States and proceeds to encode them as such.

This encoding process seems to be the psychological factor that tends to separate novice from expert. Experts seem to possess the ability to make maximum efficient use of the Encoding Principle within the cognitive system, while the novice does not either possess this talent or lacks the experience in a domain to take advantage of it. However, several experiments have been performed showing that no appreciable difference in memory capability exists between expert and novice, thus inferring that the novice has the potential of becoming an expert if he can be taught explicitly some powerful knowledge structuring techniques that would maximize the benefits from man's natural chunking ability. To illustrate how to take advantage of this chunking process, mnemonic techniques will be presented in Chapters 3 and 4.

For instance, consider the classic study on memory by Chase and Simon [Ref. 11]. In that study, it was found that chess experts, after being shown a particular board configuration for five seconds, could reproduce that configuration from memory far better than a novice. However, for the chess master to display his superior recall ability, the chess pieces had to be arranged in legitimate, meaningful positions. If the pieces were arranged in random positions, then there was no relation at all between memory of the positions and playing strength. In fact, the novice performed better at reconstructing random board positions.

Because the game of chess is thought to possess some of the mental requirements for computer programming, it did not take Shneiderman long to replicate this experiment in a programming context [Ref. 12]. Again, the findings were similar. When shown a short, structured program for a limited time period, the expert programmer was able to reproduce from memory significantly more statements from that program than was the novice. However, when a non-structured program with some of the statements shuffled in a random fashion was used for the test, the expert, again, was not able to display any superior recall ability.

The most widely accepted interpretation of these experiments attributes the superior recall to chunking. Experts have in long-term memory very efficient, hierarchically organized chunks. Access to a chunk is activated whenever the expert has encoded in short-term memory an associated retrieval cue. If these chunks and their retrieval cues can be identified, classified, organized, and taught to novices, then experts could be developed in a shorter amount of time. In the case of computer programmers, Bill Curtis writes:

> ...through experience and training, programmers are able to build increasingly larger chunks based on solution patterns which emerge in solving problems. The lines of code in the program listing:
>
> ```
>     SUM = 0
>     DO 10 I = 1,N
>     SUM = SUM + X(I)
> 10  CONTINUE
> ```
>
> would be fused by an experienced programmer into the chunk "calculate the sum of array x." The programmer can now think about working with an array sum, a single entity, rather that the six unique operators and seven unique operands in the four program statements above. [Ref. 13]

Possible programming chunks that might be taught as part of a programming class are discussed in Chapter 4.

D. SUMMARY

By understanding some of the theoretical aspects from the area of cognitive psychology that impact upon the learning process, strategies can be designed and developed that would enhance the critical Activation phase of the three-step learning process. These strategies are commonly referred to as "learning strategies" and, indeed, complete curriculums have been designed to teach students how to "learn to learn." However, the intent of this thesis is to present computer programming to novice Marine Corps programmers with the learning strategy already built into the material to be presented. The next chapter discusses some specific learning strategies.

III. LEARNING STRATEGIES

A. OVERVIEW

This chapter deals with techniques that may be employed to increase the novice's learning of computer programming concepts. Two of the techniques discussed concern (1) presenting a concrete model based on familiar knowledge, and (2) using elaboration techniques that encourage the learner to write down in his own words newly presented technical information. These two techniques can be effective when it is necessary for the student to first grasp an understanding of unfamiliar material. To enhance the retention and recall of newly presented material, a technique commonly referred to as mnemonics is also presented. Finally, the following chapter will combine the cognitive theory and these learning strategies to present some, hopefully useful, ideas to aid in the design and structuring of a new framework for the teaching of programming to novices.

B. CONCRETE MODELS

During the Recognize-Act cycle of the cognitive processor, the contents of short-term memory initiate actions associatively linked to the contents in long-term memory. In the case of novices, where domain-specific knowledge is lacking, these associative links have yet to be established. In order to satisfy the Availability condition of the three-step learning process, the learner must associate appropriate concepts already existing in long-term memory with newly received information. By the intelligent use of familiar concrete models, these prerequisite concepts residing in memory can be used to establish the necessary associative links.

To support the premise that concrete models do, in fact, enhance the learning process, several experiments have been conducted in this area. One such study discussed concerns the Brownell and Moser [Ref. 14] experiment involving the teaching of third graders the use of a subtraction algorithm.

Two groups of several hundred children were taught subtraction using two different methods. One of the groups was taught in a manner reminiscent of how computer

programmers are first exposed to programming. That is, the students were supplied with the appropriate rules (for subtraction in this case) at the beginning and given sufficient practice in the application of those rules -- in this case two-digit subtraction problems. On the other hand, a concrete model was used in the instruction of the other group of students. For this group, relatively difficult concepts, like "borrowing" and "place value," were demonstrated by the use of sticks divided into groups of ten. When the final results were analyzed, both groups of children performed equally well -- on the standard two-digit subtraction problems. However, when more complicated subtraction problems were presented, the students taught with the aid of the "bundle of sticks" concrete model displayed superior solution ability. This "transfer of learning" phenomenon is a highly desired effect and is the goal for the design of effective concrete models, especially in creative fields that possess some of the characteristics of computational procedures in mathematics, such as computer programming.

While the use of concrete models was shown to enhance meaningful learning in performing specific mathematical computations, other techniques exist to enhance the learning

of new technical information from text. For example, consider the experiment in which Bransford and Johnson [Ref. 15] presented the following text to subjects:

> The procedure is actually quite simple. First you arrange items into different groups. Of course, one pile may be sufficient depending on how much there is to do. If you have to go somewhere else due to lack of facilities, that is the next step; otherwise, you are pretty well set. It is important not to overdo things. In the short run this may not seem important, but complications can easily arise. A mistake can be expensive as well. At first the whole procedure will seem complicated. Soon, however, it will become just another facet of life. It is difficult to foresee any end to the necessity for this task in the immediate future, but then, one never can tell. After the procedure is completed one arranges the material into different groups again. Then they can be put into their appropriate places. Eventually they will be used once more and the whole cycle will have to be repeated. However, this is part of life.

When this passage was read without the benefit of just a simple title, subjects rated it as almost incomprehensible (2.3 on a 7 point scale) and, on the average, were able to recall only 2.8 out of the 18 ideas present. However, those subjects who were presented with the title, "washing clothes" before reading the passage exhibited significantly higher comprehension (4.5 on the 7 point scale) and were able to recall over twice as much material (5.8 out of the

18 idea units). But, to attain this significant improvement in assimilative and recall ability, the title had to be available before the passage was read by students. Making the title available after the passage was read did not produce an increase in performance. Other studies also have been performed in this area strongly supporting the assertion that students' recall of ambiguous and technical passages are enhanced when advanced titles, models, or sentences are given prior to reading [Ref. 15,16,17]. These techniques have become commonly known as "advance organizers."

These advance organizers tend to act as the bridge or, in the cognitive sense as the associative link, for the assimilation of new knowledge to existing schemas. Concrete models have been shown to act as extremely effective advance organizers in the learning of new scientific information. Several experiments in this area have been conducted to support this claim, also.

For instance, one such experiment was conducted by Royer and his colleagues [Ref. 18,19]. During this experiment, subjects were presented with two passages to read, one right after the other. The first article dealt with electrical conductivity and the follow-on passage dealt with the topic

of heat flow. One group was presented with concrete analogies, such as electrical conductivity being described as a chain of falling dominoes, during the reading of the first passage. The other group read both passages without the aid of the analogies. The results, again, were consistent with previous experiments. The recall of information was doubled for those students who had been subjected to the concrete analogies in the first passage. The familiar concrete models in the first passage acted as anchoring ideas and provided some prerequisite knowledge that aided learning during the reading of the second passage. More recent experiments also tend to support the notion that novices tend to learn more when the text is supported by concrete models based on familiar experiences [Ref. 20].

C. CONCRETE MODELS IN COMPUTER PROGRAMMING

While it is obvious from the preponderance of experimental evidence that specific learning strategies used in the presentation of technical information to novices significantly enhances their learning capability, convincing research on the effectiveness of concrete models in computer

programming has been done, also. In one interesting experiment, one group of subjects had the benefit of a

TABLE I

Statements Used in BASIC-like Manual

Name	Example
READ	P1 READ(A1)
WRITE	P2 WRITE(A1)
EQUALS	P3 A1 = 88
CALCULATE	P4 A1 = A1 + 12
GOTO	P6 GOTO P1
IF	P5 IF (A1 = 100) GO TO P9
STOP	P9 STOP

concrete model of a computer available before reading an instruction manual on a BASIC-like language, while the other group read the manual without the availability of such a model. Figure 3.1 shows the concrete model used, Table I describes the basic-like language, and the following passage accompanied the model of the computer:

Description of Model Provided to Subjects

The figure above represents a simple computer system which you will learn about in this experiment. The computer is made up of three main parts: (1) INPUT and OUTPUT WINDOWS, which allow communication between the computer's memory and the outside world; (2) MEMORY SCOREBOARD, which stores information in the computer; and (3) PROGRAM LIST and POINTER ARROW, which tell the computer what to do and what order to go in. Each of these three parts will now be explained.

Input and Output Window. Notice that to the far left is an input window divided into two parts. A pile of computer cards with numbers punched into them can be put in the left part of the window as the computer finishes processing each card, it puts the card on the right side of the input window. Thus when the computer needs to find the next data card it takes the top card on the left side of the input window when it is done with the card, it puts it on the right side.

On the far right is the output window. This is where printed messages (in this case only numbers can be printed) from the computer's memory to the outside world appear. Each line on the printout is a new message (i.e., a new number).

Thus the computer can store in memory a number that is on a card entered through the input window, or it can print out what it has in memory onto a printout at the output window. The statements which put the input and output windows to work are READ and WRITE statements, and each will be explained later on.

Memory Scoreboard. Inside the computer is a large scoreboard called MEMORY. Notice that it is divided into eight spaces with room for one score (one number) in each space. Also notice that each space is labeled with a name -- A1, A2, A3, A4, A5, A6, A7, A8. These labels or names for each space are called "addresses" and each of the eight addresses always has some number indicated in its space. For example, in our figure A1 shows a score of 91 and A2 has the number 17.

It is possible to change the score in any of the eight spaces; for example, the score in box A1 can be changed to 0, and you will learn how to change scores in memory later on when we discuss EQUALS statements and CALCULATION statements.

Program List and Arrow Pointer. Inside the computer to the right of the MEMORY is a place to put a list of things to do called PROGRAM LIST and an arrow which indicates what step in the list the computer should work on.

Notice that each line in the PROGRAM LIST has a number so that the first line is called P1, the second step is P2, and so on. When a program is inserted in the step, the indicator arrow will point to the first line (P1); when the first step is finished, the arrow will go to the next step on the list (P2); and so on down the list. You will learn how to control the order of steps later on when the IF statement, GOTO statement, and STOP statement are discussed. [Ref. 21]

After both groups had read the manual at their own pace, which averaged 20 to 30 minutes, the same test was administered to all subjects. This test consisted of the following six types of problems with some examples shown in Table II:

1. Generate-statement. Problems described in English that required a one-statement program for resolution.

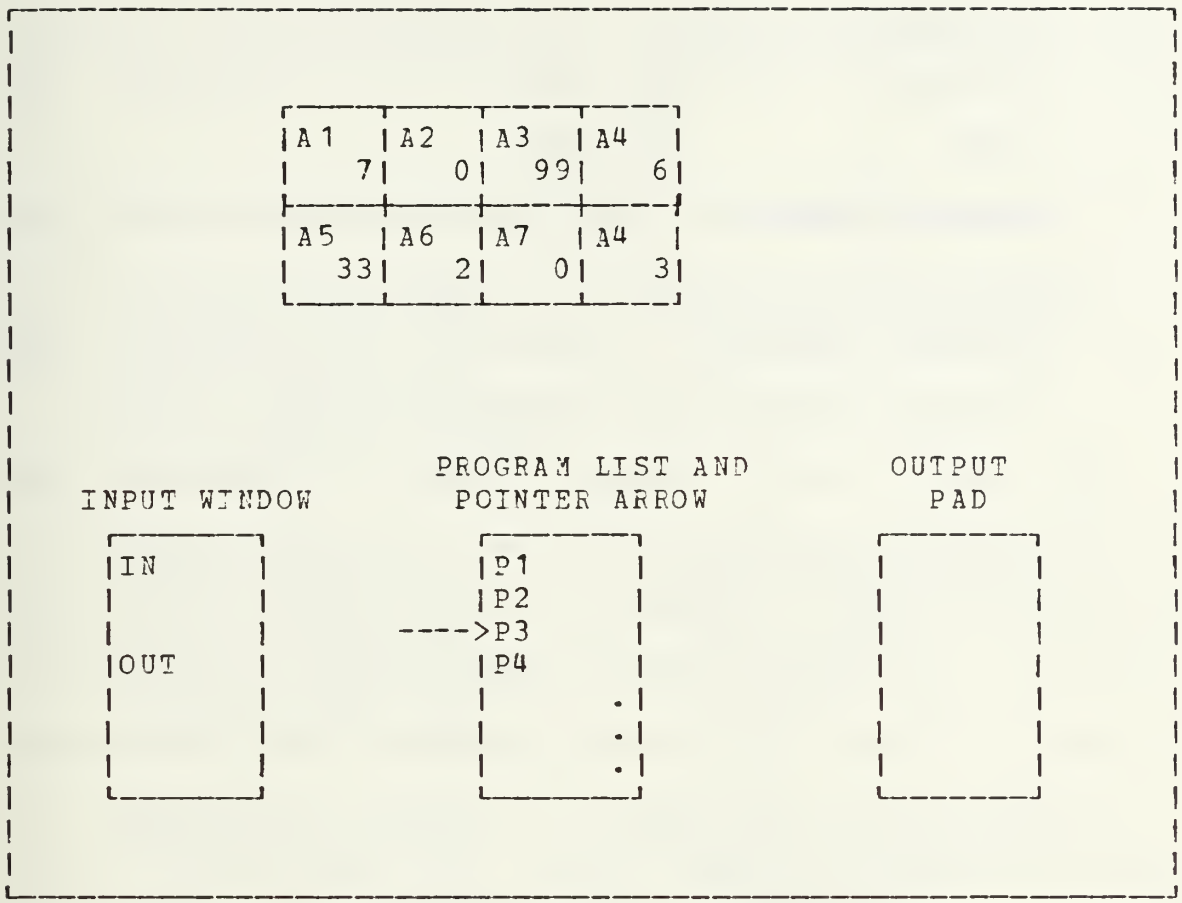

Figure 3.1 A Concrete Model of the Computer.

2. Generate-nonloop. Problems described in English and requiring a short non-looping program for the solution.
3. Generate-looping. Problems described in English but requiring a short looping program for solution.

4. Interpret-statement. A single-statement program requiring a solution written in the subject's own words.

5. Interpret-nonloop. Non-looping programs requiring an English description for the solution.

6. Interpret-looping. Looping programs requiring the English description for the solution.

The results still strongly support the argument that concrete models do enhance learning, but more importantly, the desired effect of "transfer of learning" was shown to be present in the area of computer programming. For example, the control group, the subjects who were given the material without the benefit of a concrete model of the computer, was shown to perform just as well as the model group -- but, similar to the subtraction experiment with third graders, only on those problems that paralleled material in the instruction manual, such as generate-statement and generate non-loop problems. When it came to the more complex problems where moderate amounts of learning transfer were required, such as generate-loop and shorter interpret problems, the model group performed almost twice as well. Both groups, however, did not fare well on very complex

TABLE II

Types of Test Problems for a BASIC-like Language

Generation-Statement	Interpretation-statement
Given a number in memory space A5, write a statement to change that number to zero.	A5 = 0
Generation-Nonloop	**Interpretation-Nonloop**
Given a card with a number on it as input, write a program to print out its square.	P1 READ (A1) P2 A1*A1 P3 WRITE (A1) P4 STOP
Generation-looping	**Interpretation-Looping**
Given a pile of data cards as input, write a program to print out each number and stop when it gets to a card with 88 on it.	P1 READ (A1) P2 IF (A1=88) GO TO P5 P3 WRITE (A1) P4 GO TO P1

interpret-looping problems which suggest more effective models and exercises are necessary. The actual results are given in Table III [Ref. 22].

TABLE III

Proportion of Correct Answers

Group	Generation			Interpretation		
	State-ment	Nonloop	Looping	State-ment	Nonloop	Looping
Model	.63	.37	.30	.62	.62	.09
Control	.67	.52	.12	.42	.32	.12

D. ELABORATION TECHNIQUES

Another learning strategy that possesses the potential to improve the Activation process within the cognitive system is the Elaboration Technique. In this method, students are encouraged to explain newly presented information in their own words and to relate the material to already existing knowledge. This technique also derives its theoretical power from the process of building associative links based on prior knowledge. Only this time, the learner

is actively searching for and creating his own associations. The disadvantage, however, is the difficulty in designing course material that encourages original elaboration rather than rote recitation. Several studies also have been performed to determine if the elaboration technique enhances the learning of a new computer language.

TABLE IV

An Information Management Language

Name	Example
FROM	FROM AUTOMOBILE
FOR	FOR WEIGHT IS CALLED 3000 OR MORE
AND FOR	AND FOR COLOR IS CALLED GREEN
OR FOR	OR FOR MAKE IS CALLED FORD
LIST	LIST NAME
COUNT	COUNT
TOTAL	TOTAL CURRENT VALUE
LET	LET TOTAL+COUNT BE CALLED AVERAGE

TABLE V

An Example Model Elaboration Exercise

Model Elaboration

Consider the following situation. An office clerk has an in-basket, a save basket, a discard basket, and a sorting area on the desk. The in-basket is full of records. Each one can be examined individually in the sorting area of the desk and then placed in either the save or discard basket. Describe the FOR statement in terms of what operations the clerk would perform using the in-basket, discard basket, save basket, and sorting area.

In one study [Ref. 23], the performance of two groups of subjects was compared after receiving different instructional techniques on a new programming language. The control group read an instructional manual describing an information management language (see Table IV) without the benefit of any kind of a model. The model elaboration group received an extra page that followed each page in the manual and required the student to describe the statements being learned in terms of the concrete model of a computer shown in Figure 3.1. A typical elaboration exercise is shown in

Table V and sample test problems are given in Tables VI and VII. The performances of both groups are compared in Table VIII.

TABLE VI

Sample Test Problems

Sort 1 List the owners' names for all cars weighing 3000 pounds or more.	FROM AUTOMOBILE FOR WEIGHT IS CALLED 3000 OR MORE LIST NAME
Sort 2 List the owners' names for all model green Fords.	FROM AUTOMOBILE FOR YEAR IS CALLED 1976 OR MORE AND FOR COLOR IS CALLED GREEN AND FOR MAKE IS CALLED FORD LIST NAME
Count How many cars are registered in Santa Barbara County?	FROM AUTOMOBILE FOR HOME COUNTY IS CALLED SANTA BARBARA COUNT LIST COUNT

TABLE VII

More Sample Test Problems

Compute 1 What is the average of all cars?	FROM AUTOMOBILE COUNT TOTAL CURRENT VALUE LET TOTAL/COUNT BE CALLED AVERAGE LIST AVERAGE
Compute 2 What percentage of 1977 cars are chevrolets?	FROM AUTOMOBILE FOR YEAR IS CALLED 1977 COUNT LET THIS BE CALLED COUNT 1 AND FOR MAKE IS CALLED CHEVROLET COUNT LET THIS BE CALLED COUNT 2 LET COUNT 2/COUNT 1 BE CALLED AVERAGE LIST AVERAGE

The results, again, tend to be consistent with the previous experiments. Table VIII shows that the control group did well only on the simple, retention-like problems. On the other hand, when problems required a significant amount of creative transfer, the model elaboration group displayed superior performance over the control group.

TABLE VIII

Performance Comparisons between Groups

Group	Type of Test Problem				
	Sort-1	Sort-2	Count	Compute-1	Compute-2
Model elaboration	.65	.58	.64	.64	.45
Control	.66	.64	.41	.38	.27

E. MNEMONICS

The previous two sections discussed techniques that have been shown to increase a novice's learning of technical information. This section discusses mnemonics -- encoding techniques that can increase the recall of that information. Dansereau defines mnemonics as the process of,

> ...embellishing the incoming material by creatively interrelating the items to be learned or by associating the items to a previously learned set of peg words or images (mental pictures). [Ref. 24]

According to the above definition, mnemonics, then, can include a variety of creative techniques that establish associative links between different chunks of information. The most common of these is to let the first letter of each word in a familiar phrase represent the first letter of the item to be remembered [Ref. 25]. A classic example of this technique is the phrase that is taught by Naval instructors to Navy and Marine electrician trainees to learn the following color coding scheme of resistors:

Color	Value	Multiplier
Black	0	10**0
Brown	1	10**1
Red	2	10**2
Orange	3	10**3
Yellow	4	10**4
Green	5	10**5
Blue	6	10**6
Violet	7	10**7
Grey	8	10**8
White	9	10**9

The phrase, although altered somewhat for this thesis, but still extremely effective is -- Bad Boys Ridicule Our Young Girls, But Vincent Genuflects Willingly. This easily remembered phrase allows the student to quickly reconstruct the above color coding scheme.

A more esoteric example is the phrase learned by medical students to remember the ordering of the 12 cranial nerves -- On Old Olympus, Towering Top, A Fat, Agile German Vaults And Hops. Of course, those 12 cranial nerves, in order, are easily recalled -- olfactory, optic, oculomotor, trochlear, trigeminal, abducens, facial, acoustic, glossopharyngeal, vagus, accessory, hypoglossal.

This thesis will use a variation of the above technique. Instead of using the "first-letter" mnemonic technique illustrated in the two examples above, meaningful acronyms will be designed based on associative links created from common military experiences whenever possible. While the following chapter discusses this method in detail, a brief example is the previously discussed acronym, "BRASS." Here, the acronym BRASS is composed of a mnemonic technique that takes the first letter from each of the marksmanship principles, and, coincidentally, happens to form a marksmanship-related word -- brass. Now, suppose a separate course of instruction covered material, say, on marksmanship principles during a frontal assault or attack, rather than at the rifle range. An acronym might be contrived that would relate BRASS to attack, such as BRASS ATTACK, or BRASS TACKS. This hypothetical example illustrates the approach

taken to form an associative link between two chunks of related instruction that would enhance retention and recall.

The contrived example above brings up what might seem like two synonymous ideas -- that is, the idea of a mnemonic technique such as acronyms, and the concept of "chunking," or chunks. In this thesis, the concepts are not the same. The mnemonic techniques create retrieval cues and advance organizers for a complete schema, or chunk, of information. The process in which a retrieval cue is associatively linked, or fused, to its schema is referred to as "chunking." This process, when followed in course design, is referred to in this thesis as Cognitively Designed Instruction.

F. SAMPLES OF COGNITIVELY DESIGNED INSTRUCTION

Examples of the power of mnemonic techniques were illustrated in the previous section. This section references some training materials cognitively designed to make use of the other learning strategies, such as analogies, comparisons, and elaboration techniques. These examples are presented to provide a flavor for the next chapter's presentation of suggested instruction based on these ideas.

Appendix A illustrates the use of associative links based on previous knowledge to enhance the assimilation and retention of scientific information.

Appendix B illustrates a technique that proves to be helpful in memorizing long lists of items. However, this technique, referred to as "imaging," is not explicitly discussed in this thesis. The reason being that any lists important enough in the software engineering curriculum to be committed to memory will use mnemonics explicitly developed by the instructor and shown to the student.

IV. SUGGESTED LEARNING STRATEGY APPLICATION

A. SCOPE

While the effectiveness of a computer programmer obviously depends on his command of a particular programming language, too often an undisciplined application of this command can, and usually does, contribute to major problems during the life-cycle of software projects. In much the same way as a Marine rifle platoon commander is governed by a set of fire-control principles in the employment of his fire-power, the Marine computer programmer needs to be governed by a set of programming-control principles in the employment of his programming-power.

However, while the rifle platoon commander's set of tools has been constantly updated and augmented with the newest developments in hardware technology (TOWs, DRAGONs, LASERs, etc.), the Marine programmer is required to accomplish his mission using antiquated, second-generation programming languages, such as COBOL, and low-level systems programming assembler languages, such as IBM's BAL.

But the intent of this thesis is not to argue the need for a gradual "retooling" of Marine Corps software, rather, the intent is to recognize the need for a new, cognitive approach to training in the neglected area of computer programming. This new approach is designed to transcend a particular programming language and instill in the programmer an instinctive discipline of programming. Although he will know the programming language he is using is not state-of-the-art in software technology, he will be able to employ the language in a disciplined manner, taught and learned based on the cognitive principles and learning strategies introduced earlier. This approach will ensure the maximum effectiveness possible from any language.

To introduce these ideas, this chapter is divided into several sections. One section focusses on the goals of programming that every programmer should have committed to memory and available for instinctive recall. The following section will present the most up-to-date software engineering principles that good programmers should follow to achieve the programming goals. Another section will deal with the teaching of difficult concepts usually encountered when students are learning to program for the first time.

But, more importantly, embedded in the presentation of this material will be some of the learning techniques based on the cognitive theory presented in an earlier chapter. Whenever the technical material presented to the student requires understanding first, then concrete models or analogies will be used anchored in the prerequisite knowledge of familiar military experiences. Whenever the material needs to be organized in an efficient manner for increased retention and efficient recall then meaningful mnemonic aids will be designed and proposed to maximize the effect of the Encoding Principle. By explicitly building these learning techniques into the material, a more systematic and effective programming process might become second-nature, or instinctive, for Marine programmers.

B. THE GOALS OF PROGRAMMING -- NO "RUME" FOR ERROR

This thesis proposes that the specific goals of programming should be taught explicitly and concurrently with the initial instruction of the programming language being learned by the novice. This approach is motivated by the success of the experiment discussed earlier concerning the topics of electrical conductivity and heat flow. In

that experiment, students were able to learn more on heat flow when they first read a passage on electrical conductivity containing meaningful concrete analogies that acted as anchoring ideas for the learning of concepts in heat flow. Similarly, it is proposed that by providing cognitively designed instruction on the goals of programming concurrently with the beginning of a course of instruction in a programming language, crucial prerequisite knowledge is made available. The student then should be able to make, or will be shown, important associative links between these desirable goals and the new programming language's inherent capabilities to achieve these goals.

This cognitively designed instruction on programming goals is introduced with a learning strategy built in. That is, an advance organizer, in the form of a simple, meaningful title, "No RUME for Error," is introduced and explained. This particular advance organizer possesses additional power to enhance recall and retention through the use of an organizing acronym, "RUME," that creates a retrieval cue in much the same way as did "BRASS" discussed in the first chapter. This time, though, RUME represents:

R -- Reliability
U -- Understandability
M -- Modifiability
E -- Efficiency

Consider the theoretical power and characteristics this one simple title possesses to enhance the learning of the programmer. First of all, this particular course of instruction is presented at about the same time the student has submitted his first simple program for compilation. The resulting list of syntax errors from this initial experience will impress upon the student that in programming there is "no room for error." Of course, a simple modification of the conventional spelling of the word "room" to a phonetically equivalent acronym "RUME" further intensifies the associative link being created during this experience. After the substitution has been made and noted -- RUME for "room" -- then the acronym can be broken down into the individual goal each letter represents and further discussed. After the presentation of these four programming goals -- reliability, understandability, modifiability, and efficiency -- the programmer has a concise, complete, organized set of goals available instinctively during the programming process by reminding himself that there is "no RUME for error" -- a thought programmers constantly have anyway, especially when COBOL is being used.

In preparing the training aids to present the individual goals of the RUME chunk, an important consideration surfaces based on one of the four principles of operation for the model human processor listed in Chapter 3 -- the Discrimination Principle. This principle stated that the difficulty of memory retrieval is determined by the number of candidate chunks that exist in memory similar to the active retrieval cue [Ref. 26]. Therefore, in situations where the technical material possesses rather straightforward concepts, as do the individual RUME goals, it seems, theoretically, better not to devise unnecessary acronyms that might interfere with the main acronym or advance organizer. This assertion is in consonance with the Discrimination Principle.

This is not to imply, however, that organizing nested chunks of information by the use of multiple acronyms should not be considered if the characteristics of the information should make them necessary. Indeed, in some instances, the material is so crucial, chunks can be organized by numerous acronyms to achieve a powerful cascading effect where one chunk of information has embedded retrieval cues for other chunks which, in turn, possess acronyms that further activate related chunks of information.

As a classic, practical example, note the reliance on nested chunking, achieved by multiple acronyms, used in Marine Corps tactical training. In the small unit leaders Troop Leading Steps (a retrieval cue of BAMCIS) in Figure 4.1 paragraph 4 leads to the activation of the 5 Paragraph Order chunk (a retrieval cue of SMEAC) illustrated in Figure 4.2 [Ref. 27]. Embedded acronyms also appear within each major chunk linking together related chunks of information. Specifically, the deepest level occurs in paragraph 1b of Figure 4.1 where the "B" in BAMCIS (B -- Begin Planning) triggers the METT chunk, and the "E" in METT (E -- Enemy Situation) activates the SALUTE and DRAWD acronyms.

However, as the chunks are nested deeper and deeper, meaningful associative links become more difficult to establish among all the acronyms. The Discrimination Principle says that practice time to learn the material must, therefore, also increase. This situation brings up the crucial design question that must be resolved in devising cognitive instruction. Does the importance of the material to be learned merit the time required to design meaningful cognitive instruction to enhance the long-term retention and recall of that material?

TIPS FOR SMALL UNIT LEADERS

This card contains two items which every unit leader must master.

The Troop Leading Steps are a guide for the thoughts and actions of the unit leader from the time he receives his attack order until he successfully accomplishes the mission.

The 5 Paragraph Order must be well formulated, follow the listed sequence, be concise, confidence-inspiring, and forceful in tone.

AFTER RECEIPT OF ATTACK ORDER FROM HIGHER AUTHORITY BEGIN:

TROOP LEADING STEPS

1. BEGIN PLANNING
   a. Plan use of available time. (Reverse Planning, half-rule).
   b. Formulate your estimate of the situation based on METT.
      M - Mission assigned to your unit.
      E - Enemy situation (SALUTE) and capabilities (DRAWD).
      T - Terrain and weather: examine (KOCOA) referring to your map.
      T - Troops and fire support available (friendly)
      Formulate a preliminary plan of attack based on mental estimate of the situation. (formation, form of maneuver, fire support plan).

2. ARRANGE FOR
   a. Movement and readiness of your unit by issuing a warning order to subordinates (when, where, what, how) and notifying them of time and place (vantage point) to receive the operation order.
   b. Reconnaissance (if visual recon is possible)
      (1) Select a route for recon (when, where, how).
      (2) Personnel to accompany your on recon (subordinates, adjacent, or supporting unit leaders).
   c. Coordination (meeting with adjacent and supporting unit leaders).

3. MAKE RECONNAISSANCE (may be map or visual recon)
   a. Confirm your tactical control measures.
   b. Analyze terrain and revise earlier estimate of situation as necessary (use METT concentrating on KOCOA).
   c. Finalize coordination with adjacent and supporting unit leaders.

4. COMPLETE PLAN
   a. Receive recommendations from your staff or supporting unit leaders.
   b. Based on your estimate of the situation, and having examined all the courses of action, arrive at a decision as to the plan of attack and write operation order using SMEAC.

5. ISSUE ORDER
   a. Issue your operation order including an orientation.

6. SUPERVISE
   a. The planning and preparation by subordinates.
   b. The conduct of operations and accomplishment of the mission.

Figure 4.1 The BAMCIS Troop Leading Steps.

Although the RUME goals are very important for a programmer to be thinking of constantly during the programming process, and the "No RUME for Error" advance organizer enhances the retention and recall of the aim of these goals, it is not necessary to develop acronyms for nested organizing chunks when presenting instruction on the individual goals. The following key points, though, should be included in any further course of instruction that might be developed.

1. Reliability

The goal of reliability is to prevent failure in conception, design, and construction. Furthermore, reliability means the graceful recovery from failure in operation or performance. Reliability is a characteristic that can only be built in from the start and not added on at the end as part of a maintenance activity. [Ref. 28]

2. Understandability

To achieve understandability, it is not enough for the program to be readable, although readability is certainly a necessary condition. Rather, the entire conceptual structure must be considered. The structure must

5 PARAGRAPH OPERATION ORDER

ORIENTATION - Always give an orientation to include your location, objective, direction of movement, tactical control measures, terrain, and vegetation.

1. SITUATION

a. Enemy Forces: SALUTE and DRAWD
b. Friendly Forces: HAS; Higher unit's mission, Adjacent unit's location, Supporting unit's mission.
c. Attachments and Detachments: Units attached or detached and the effective time.

2. MISSION

a. A clear, concise statement of the mission of your unit. Who, What, When, and Where, Why (5 W's)

3. EXECUTION

a. Concept of Operations: A brief concept of your unit's plan of attack explaining HOW it is going to accomplish the mission. (Must include your unit's formation and form of maneuver).
b. Mission of organic units: Detailed explanation of actions required for each subordinate organic unit from the line of departure through consolidation.
c. Mission of attached units: Only applicable to platoon-sized units and above.
d. Mission of reserve units: Only applicable to company-sized units and above.
e. Coordinating instructions: Information of a tactical nature common to two or more subordinate units, to include tactical control measures (Final CL, LD, frontages, base unit, time of attack, etc.)

4. ADMINISTRATIVE AND LOGISTICS

a. Rations, ammunition, medical POW's etc. (4 B's)

5. COMMAND AND SIGNAL

a. Signal: Signal identification and instructions (every primary signal must include alternate)
b. Command: Your location and next higher unit commander's location.

ACROYNMS

ENEMY SITUATION	ENEMY CAPABILITIES	TERRAIN ANALYSIS
S - Size	D - Defend	K - Key terrain
A - Activity	R - Reinforce	O - Observation and fieldss of fire
L - Location	A - Attack	C - Cover and concealment
U - Unit	W - Withdraw	O - Obstacles
T - Time	D - Delay	A - Avenues of approach
E - Equipment		

Figure 4.2 The 5 Paragraph Order.

be represented in a clear, consistent notation. Whenever assembly language is used in an application, the tradeoff is efficiency for program understandability. [Ref. 29]

3. Modifiability

Modifiablity implies ease of maintenance, and maintenance costs on software sometimes exceed fifty percent of the total life cycle cost. However, this is the hardest goal to achieve -- especially with second generation languages that make it hard to apply modern software engineering principles. Modifiability requires the ability to have an adaptable, evolutionary design that employs some standardized software building blocks. [Ref. 30]

4. Efficiency

Last, but certainly not least, and usually prematurely put first, is efficiency. Obvious inefficiencies are unacceptable, but all efficiency issues should be weighed against the other goals. For example, achieving the goal of modifiability can provide the means to achieve the desired efficiency during the tuning phase of software development. The point is, the efficiency goal should not dominate the programming project but kept in context with the other goals. [Ref. 31]

C. ACHIEVING THE GOALS -- THE "AIM FOR LUCC" PRINCIPLES

Certainly, the attainment of the PUME goals[1] is not achieved by aiming for luck as the section title might suggest. However, the title is another advance organizer that will be subordinate to the "No RUME for Error" root and its associated chunk of information. In designing a retrieval cue for this new chunk, emphasis was placed on creating a concrete analogy that would take advantage of the students' military background. Based on theory and experiments presented in earlier chapters of this thesis, concrete analogies devised in this manner should create a more vivid, lasting associative link between nested chunks of information. Although the BAMCIS (Troop Leading Steps) and the SMEAC (5 Paragraph Order) acronyms must rely on significant practice to create the link between them, this thesis attempts to design retrieval cues for related chunks of information with meaningful associated acronyms that help activate the recall of other chunks naturally, without rote memorization and with little practice time. As an example

---

[1] Note the added benefit of organizing chunks is being able to reference a whole block of instruction or ideas by the simple mnemonic -- RUME. The student now has a concise, organized schema of the programming goals in memory ready for the further efficient assimilation of new information.

of this particular cognitive approach, consider an analogy designed to aid the student in associatively linking the "No RUME for Error" goals to the software engineering principles available to achieve those goals.

First of all, the student is now familiar with the important programming goals -- concisely organized and instinctively ready for recall with the "No RUME for Error" retrieval cue. The software engineering principles to achieve those goals are presented in the form of another advance organizer entitled, "The AIM for LUCC" principles. The embedded acronyms -- AIM and LUCC -- are then broken down for the student into the respective software engineering principle each letter represents:

A -- Abstraction
I -- Information Hiding
M -- Modularity
for
L -- Localization
U -- Uniformity
C -- Consistency
C -- Confirmability

Even though these seven principles may, at first, seem formidable for a novice programmer, the student can be assured that each one will be explained in a manner that he

can understand. But, for the time being, he at least knows that the "AIM for LUCC" principles are in direct support of the "No RUME for Error" goals. Again, the student should notice that "LUCC" is not the conventional spelling for "luck," a minor detail that was shown in Appendix A to be an advantage for recall rather than a detriment. But, the acronym LUCC will now be spelled "LUCK" to make the "AIM for LUCK" retrieval cue more meaningful and even easier to retain.

Even though the phrase "direct support" mentioned above subtly possesses a military connotation, a stronger connection, based on more vivid, recent military experiences is suggested to make a permanent associative link between the RUME acronym and the very important "AIM for LUCK" principles. After all, the goal is to make these modern software engineering principles instinctive to the Marine programmer. The suggested analogy draws upon the students' intensive military training and experience in the area of marksmanship. According to the cognitive theory, this prerequisite knowledge will act as an anchoring mechanism so that a lasting associative link can be established between the RUME chunk and the "AIM for LUCK" chunk.

The analogy proceeds in the area of one of the most grueling experiences in boot camp for Marines -- the two week training period at the rifle range. During these two weeks, the Marine spends many hours in classroom instruction on marksmanship principles. He then spends many hours at the "snapping-in" range applying this instruction in practice. His hard work will be rewarded if he can score a total of 220 out of a possible 250 points on qualification day. This score would earn the Marine the highest and most prestigious award usually possible in boot camp -- the coveted rifle expert badge.

However, to attain the designation of rifle expert, the shooter must fire 25 rounds from 100 yards from three different shooting positions; 15 rounds from the 300 yard line from two different positions; and the last 10 rounds from 500 yards from the prone position. Each round is worth 5 points for a bullseye, 4 for the 4-ring, 3 for the 3-ring, 2 for the 2-ring, and 0 for a miss. Shooters quickly learn to keep a mental running score on how many points they have "dropped." For example, a 10-round string of rapid fire is worth 50 possible points if all rounds hit the bullseye. If a shooter has all bulls but one round in the 4-ring then he is shooting a possible 249 at this point, providing he has

not dropped a point up until this time. But, by the time the shooter gets back to the 500 yard line, he usually knows exactly how many pcints he can drop and still earn the expert designation. Sometimes, he must shoot close to all bullseyes with his final 10 rounds from the 500 yard line.

Now, the associative link can be made. Marines know that at a distance of 500 yards, the slightest error in sight alignment, sight picture, or any other "BRASS" marksmanship principle can move a round completely off the target, not to mention the bullseye. Clearly, there is no room for error. Moreover, to make things worse, now an unexpected gust of wind can have a devastating effect on where the round will impact downrange. Thus, the Marine must aim and hope for luck that the wind remains steady.

By using this very common experience that all Marines have shared at one time or another, the "No RUME for Error" programming goals chunk is vividly linked to the "AIM for LUCK" principles. Theoretically, the recall and the associated instruction that goes with each chunk should be greatly enhanced.

D. THE PRESENTATION OF THE AIM FOR LUCK PRINCIPLES

The goals of programming and the software engineering principles to achieve those goals have now been cognitively designed for efficient storage in long-term memory by the student. Cognitively designed retrieval cues also have been explicitly taught and associatively linked to cause a natural, cascading effect for the recall, or activation of this information. The next phase is to introduce each individual principle in the AIM for LUCK chunk in a similar, cognitive manner where appropriate.

However, while the RUME goals were relatively straightforward information, the AIM for LUCK principles will be the anchoring chunk providing the necessary prerequisite knowledge for which any programming language can be taught. For example, when introducing one of these principles to the student, familiar models or analogies will be designed so the student grasps a firm understanding of the concept first. Then, specific examples, <u>in the programming language being learned</u>, will be presented to illustrate the capability of that language to support the particular principle being discussed. It is here where the inherent weaknesses of a particular language will be

revealed. Then, techniques can be shown that are designed to minimize the effects of those weaknesses. This overall approach should also ease the future transition from second-generation languages to more sophisticated languages specifically designed to enforce the AIM for LUCK principles, such as the DOD-backed ADA programming and design language.

As an example of the overall approach, consider the course of instruction that could be designed to introduce the Abstraction Principle. A definition that possesses a rather high level of sophistication describes abstraction as, "...the essence is the extraction of essential properties while omitting inessential details" [Ref. 32]. Keeping in mind that the instruction is targeted at Marine programmers recently graduated from high school, the idea of the Abstraction Principle can be explained at a lower, but just as effective, level of sophistication. The approach used is an analogy based on military experience from which every student can personally identify.

The analogy appears in a story about a young second lieutenant taking his final examination on leadership. He is giving a hypothetical problem in which he is charge of putting up a 20 foot flagpole and has at his disposal a

sergeant and 12 young enlisted Marines to help him accomplish the mission. Invariably, the young lieutenant begins his BAMCIS troop leading steps and begins his planning. He gets together his calculator, and drafting equipment and begins drawing up elaborate plans using trigonometric functions to make sure this flagpole is erected perfectly. Predictably, though, he runs out of time to finish the test and gets a low score. However, in his eagerness to find out the solution, he rushes to the instructor and demands the solution arguing that there just wasn't enough time for a question like that. The instructor tells the young lieutenant that the answer was simple -- "Sergeant, put up that flagpole!"

It would be hard to argue against this analogy as a classic, simple method to define abstraction. The point is made that the lieutenant's responsibility is to get the flagpole up. However, the details in accomplishing the goal are subordinate to the lower levels -- in this case, the squad leader -- in the programming sense, subroutines, functions, or purposeful modularity. At this point, specific programming language examples of subroutines, functions, or any other technique that exemplifies the Abstraction Principle could be presented.

As another example, consider the Information Hiding Principle and how its meaning can be conveyed to young Marine programmers. First of all, consider the esoteric definition assigned to information hiding by Ross, "Hiding is concerned with defining and enforcing access constraints that, without the hiding principle, would only be implicit in some purpose, concept, mechanism, or usage description."

But, what does this important principle really mean in terms understandable to a novice Marine programmer? One possible analogy lies in the functioning of the Marine's chain of command. For example, if a fire team member in a squad has a problem, he must strictly follow his chain of command to seek a solution. He must first go to his fire team leader. If the fire team leader does not possess the resources or power to solve the problem, then the fire team leader would bring the problem to his squad leader. The problem is only elevated to the level that is capable of solving the problem. The higher echelons need not be concerned of everyone's problems if they can be resolved satisfactorily at the lowest possible level.

Information hiding is analogous to the chain of command in so much as its purpose is to make inaccessible certain details that should not affect other parts of a system. The

Uniform Code of Military Justice acts as the mechanism to enforce the adherence to the chain of command while in a programming language, strong typing facilities or private data types, such as Ada's, enforces the information hiding principle.

E. SYNTAX AND SEMANTICS -- "AID OF THE DICE"

The two highest levels of a cognitive type framework for software engineering, with an emphasis on programming, has been presented. It was discussed how the "no FUME for error" goals supported by the "AIM for LUCK" principles organizes the necessary information for disciplined programming into easily remembered associated mnemonics, or retrieval cues. Now what is needed is an easily remembered advance organizer that interfaces these goals and principles with the actual syntax and semantics of a particular programming language. The suggested mnemonic is "AID of the DICE" and is very easily associatively linked to the AIM for LUCK principles through the common link of LUCK and DICE.

The "AID of the DICE" syntax and semantics organizer is the framework that will allow the specific syntax and semantics of the programming language to be presented. The mnemonic decomposes into:

A -- Assignment statements
I -- Identifiers
D __ Data types

of the

D -- Data structures
I -- Input/Output
C -- Control statements
E -- Expressions

This simple mnemonic will allow specific features of the programming language to be introduced in any order the instructor deems necessary. Conventional teaching techniques could be used to present the syntax, but some analogies could be devised to present the more difficult concepts, along with concrete models of the computer that were presented earlier. The important point is the advance organizer provides the infrastructure to efficiently organize the material to be presented to the student.

After each topic in the "AID of the DICE" acronym is presented and understood by the student then the programming chunks discussed in Chapter 2 could be presented. For example, consider again the "calculate the sum of array X" program chunk:

```
      SUM = 0
      DO 10 I = 1,N
      SUM = SUM + X(I)
   10 CONTINUE
```

This particular chunk is composed of assignment statements, identifiers, data types (implicit in this case), a data structure (the array X), a control statement, and an expression. By adding a PRINT SUM statement at the end, every topic in the AID of the DICE acronym is illustrated. More importantly, this particular chunk could be studied and more chunks could be realized through variations. For instance, by adding another expression, such as AVERAGE = SUM/N, after the continue statement, another chunk is formed with a retrieval cue of "calculate the average of array X." Moreover, other programming chunks could be collected and presented as part of a "tool kit" for the novice programmer. For example, chunks that perform sorting, searching, swapping, string manipulation, or linking functions could be explicitly taught.

F. SUMMARY

This chapter presented a small sampling of cognitively designed materials that could aid the Marine Corps programmer in assimilating and retaining software engineering skills. The cognitive infrastructure, formed by

the RUME goals supported by the AIM for LUCK principles that guide the implementation of the AID of the DICE syntax and semantic topics, attempts to provide a meaningful structure that is capable of enhancing the further assimilation of programming knowledge as the programmer acquires more experience. The thesis being that this assimilation is more rapid and longer lasting due to the efficient storage and meaningful retrieval cues of the software engineering goals and principles.

V. FUTURE RESEARCH AND CONCLUSIONS

A. FUTURE RESEARCH

One area of future research concerns the complete development of a core of programming chunks that could be taught to entry-level programmers under the AID of the DICE syntax and semantics organizer. A proposed method to accomplish the establishment of a core of chunks is to solicit the expert programmers in the Marine Corps data processing activities for those chunks of code that reoccur from application to application and are considered important. This direct input from the field would complement some of the basic chunks mentioned earlier. After these chunks are identified, collected, and classified they could be integrated into the current instructional system on the language concerned. The current instructional system would then also be reviewed for possible enhancements using the cognitive approach discussed earlier in this thesis.

For mid-career Marine programmers, the level of sophistication could be raised to a higher level. This means sophisticated and more detailed training materials that, still, would be introduced under the RUME goals and the AIM for LUCK principles. However, instead of bringing the mid-career programmer to the Computer Science School in Quantico, Virginia, a traveling team of instructors would visit each major data processing activity teaching the modern software engineering curriculum. This particular method has been proven successful by the current Systems Analysis and Design team that travels to the various commands.

B. CONCLUSIONS

This thesis has attempted to provide an easily remembered structure for the software engineering process at a level of sophistication compatible with Marine Corps entry-level programmers. This structure was designed so that the programmers could have readily available in their thinking process modern software engineering goals and principles that ultimately affect the quality of software. In short, during the process of creating programs, an

instinctive reminder of programming discipline is instilled in one simple retrieval cue -- the AIM for LUCK principles.

Also suggested at the lower level of the overall structure was the AID of the DICE syntax and semantics organizer. This particular framework serves as an advance organizer for which specific programming language features could be introduced. Besides the conventional teaching methods used in presenting language syntax and semantics, this structure can act as an organizing mechanism for the introduction of various, useful programming chunks that would start the novice programmer on his quest to becoming an expert.

This overall learning strategy approach to programming, theoretically, should enhance the programmer's assimilation and retention of information. Numerous experiments and studies have been referenced that tend to support this assertion.

However, future research is needed to fully develop a software engineering curriculum with an emphasis, for the Marine Corps, on programming. The initial approach could be to develop selected portions of this curriculum that could be taught to generate feedback. This feedback could then be

used to determine if the learning strategy approach to software engineering is worth the effort of a full scale implementation effort.

APPENDIX A

SAMPLE INSTRUCTION 1

How Do the Arteries Differ from the Veins?

Arteries and veins are both hollow tubes through which the blood flows. The two differ, however, in three important respects. The work of an artery, first of all, is to carry blood rich in oxygen from the heart to to the various organs of the body. The work of a vein, on the other hand, is to return to the heart blocd laden with carbon dioxide. In structure, too, veins and arteries differ markedly. Although the walls of both are composed of three coats of tissue, the veins are thinner than the arteries and less elastic. A third point of difference is the way the blood moves within the two types of vessels. Propelled by the force of the heartbeat, the blood rushes through the arteries, which expand and contract to push it forward in spurts. In the veins, however, the blood flows slowly and smoothly.

Directions. I want you to learn the information contained in this paragraph and most importantly, I want you to develop learning aids to study this information. You must learn to distinguish between the veins and the arteries. To help you remember this fact you might try to form a picture in your mind of a thin hollow tube when you think of a vein. Or you might make up a sentence or story which would associate a vein with a thin tube such as the vain woman was thin as a rubber tube. Although the word vain in this sentence is not the same as the word

vein meaning a structure in the body it could still help you to learn this property of veins.

Concentrating on pictures or images we form in our minds can be a powerful aid to our memory; so can forming sentences or little stories which help us to remember information we must learn. These are both different ways of trying to make new or unfamiliar material more meaningful to us so that it will be easier to learn.

You can also try to relate the information contained in the reading to something else you already know. Then try to figure out as many ways as you can that the two are related or similar. For example, a vein is like a thin rubber water pipe. Both are thin tubes, relatively rigid and have fluid going through them.

I would like each of you to read this passage carefully and try to create some learning aids that you think will help you to learn the different properties of arteries and veins. [Ref. 33]

APPENDIX B

SAMPLE INSTRUCTION 2

Home Economics Shopping List

Crackers
Tomatoes
Chicken
Pickles
Soup
Lettuce
Tea
Mustard
Ice
Bread
Salt

Directions. This is a shopping list for a home economics class. Imagine your are in the class and you must remember this list when you get to the grocery store. Assuming you cannot take the list with you, how could you remember it? The order is not important but you must remember all the items.

In learning a list of items, it is helpful for a student to have some way of associating all of the items in the list. One way to do this is to create a little scene or story that would include each on the items. For example, to remember a list of school supplies such as pencil, paper, and textbook, you might imagine a student reading a textbook and using her pencil to take notes. Remembering the image or story would help you to remember the articles on the list. Making up a story is one way to help someone learn this list. See how many different learning aids you can create that would help

you. Remember, you do not have to remember the items in order, but you must remember all of them.

Also notice that for different materials some types of aids are better than others. Try to come up with stories or images that would help you to connect all the words so that if you forget one of them the logic or theme of the story could help you to remember it. [Ref. 34]

LIST OF REFERENCES

1. Neisser, U., Cognitive Psychology, pp. 4, Meredith, 1967.

2. Bransford, J.D., Human Cognition, Wadsworth, 1979.

3. Mayer, R. E., "The Psychology of Learning Computer Programming by Novices," Computing Surveys, v. 13, pp. 121-141, March 1981.

4. Card, S. K., Moran, T. P., and Newell, A., The Psychology of Human-Computer Interaction, pp. 24-33, Lawrence Erlbaum Associates, 1983.

5. Ibid.

6. Mayer, "Psychology of Learning Computer Programming."

7. Ibid.

8. Card et al.

9. Ibid.

10. Ibid.

11. Chase, W. G., and Simon, H. A. "Perception in Chess," Cognitive Psychology, v. 4. pp. 55-81, April 1973.

12. Shneiderman, B. and Mayer, R. E., "Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results," International Journal of Computer and Information Sciences, v. 8, pp. 219-238, September 1979.

13. Curtis, W., "Fifteen Years of Psychology in Software Engineering: Individual Differences and Cognitive Science," IEEE, v. 97, pp. 97-106, January 1984.

14. Brownell, W. A. and Moser, H. E., "Meaningful vs. Mechanical Learning: A Study in Grade III Subtraction," Duke University Research Studies in Education, pp. 1-207, Duke University Press, 1949.

15. Bransford, J. D., and Johnson, M. K., "Contextual Prerequisites for Understanding: Some Investigations of Comprehension and Recall," Journal of Verbal Learning and Verbal Behaviour, v. 11, pp. 717-726, 1972.

16. Dooling, D. J., and Lachman, R., "Effects of Comprehension on the Retention of Prose," Journal of Experimental Psychology, v. 88, pp. 216-222, 1971.

17. Dooling, D. J., and Mullett, R. I., "Laws of Thematic Effects on Retention of Prose," Journal of Experimental Psychology, v. 97, pp. 404-406, 1973.

18. Royer, J. M., and Cable, G. W., "Facilitating Learning in Connected Discourse," Journal of Educational Psychology, v. 67, pp. 116-123, 1975.

19. Royer, J. M., and Cable, G. W., "Illustrations, Analogies, and Facilitative Transfer in Prose Learning," Journal of Educational Psychology, v. 68, pp. 205-209, 1976.

20. Mayer, R. E., and Cook, L., "Effects of Shadowing on Prose Comprehension and Problem Solving," Memory and Cognition, v. 8, pp. 98-104, 1981.

21. Mayer, R. E., "Different Problem-solving Competencies Established in Learning Computer Programming with and without Meaningful Models," Journal of Educational Psychology, v. 67, pp. 725-734, 1975.

22. Mayer, "Psychology of Learning Computer Programming."

23. Mayer, R. E., "Elaboration Techniques that Increase the Meaningfulness of Technical Text: An Experimental Test of the Learning Strategy Hypothesis." Journal of Educational Psychology, v. 72, pp. 770-784, 1975.

24. O'Neil, H. J., editor, Learning Strategies, pp. 5-6, Academic Press, 1978.

25. Ibid.

26. Mayer, "Elaboration Techniques."

27. Marine Corps Development and Education Command, Tips for Small Unit Leaders, Quantico, Virginia, 1978.

28. Freeman, P., and Wasserman, A. I., editors, Tutorial on Software Design Techniques, pp. 54-64, IEEE, 1980.

29. Ibid.

30. Ibid.

31. Ibid.

32. Ibid.

33. O'Neil, pp. 45.

34. Ibid, pp. 45-46.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Technical Information Center Camden Station Alexandria, Virginia 22314	2
Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
Computer Technology Programs Code 37, Naval Postgraduate School Monterey, California 93943	1
Dr. Gordon Bradley Code 52BZ Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
Robert Westbrook Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
Capt. Donald D. Begley Computer Science School Marine Corps Development and Education Command Quantico, Virginia	4
LCDR Ron Kurth Code 52kh Department of Computer Science Naval Postgraduate School Monterey, California 93943	1